Practical

Computational Intelligence.

1) Design a distributed application using RPC for remote computation where client submits an integer value to the server and server calculates factorial and returns the result to the client program.

→ Objective:
The goal is to demonstrate Remote Proddure Call (RPC) in python using XML-RPC where:

- The client sends an integer to the server.
- The server computes the factorial.
- The server returns the result back to the client.

→ What is distributed computing?

- Distributed computing refers to a model where different components of a software system are spread

across multiple machines, which work together to acheive a common goal.
- Each machine in the system can act as a client, server or both.
- Key benifit: Task can be split among different systems for scalability, performance and resource sharing.

→ What is RPC (Remote Procedure Call?)
- RPC is a protocol that allows a program to execute a function/ method on another computer (server), just like calling a local function.
- The function runs on the remote machine and the results are sent back to the caller.
- RPC abstracts away the underlying network communication details, giving a feel of calling a local function.
- RPC supports both synchronous and asynchronous modes.

→ What is XML - RPC?
- XML-RPC is a lightweight RPC protocol that uses:
  - XML for encoding messages

(function calls and responses).
- HTTP as the transport protocol.
- XML-RPC is simple and platform-independent.
- Python has built-in support via:
  - xmlrpc.server
  - xmlrpc.client

→ Client-Server Architecture:
- Server:
  - Runs an XML-RPC server on localhost:8000.
  - Registers a factorial(n) function.
  - Listens for requests and returns results to client.
- Client:
  - Connects to the server using ServerProxy.
  - Sends a number.
  - Receives and displays the factorial.

→ How RPC communication Happens.
1. Client calls a function via a proxy object.
2. The function call is converted to XML and sent over HTTP POST.
3. Server receives it, decodes the ~~XML~~ XML, maps it to the registered function.

4. The function is executed, result is encoded in XML and returned over HTTP.
5. Client decodes the XML and gets the result.

This is all abstracted under the hood - you write it like a normal function call.

→ Error handling :-
- The server checks if input is a non-negative integer. If not, it raises a valueError. Errors are logged in rpc-server.log. The client catches exceptions if the server is unreachable or input is invalid.

→ Logging on the server :-
- Server uses the Logging module to write information to rpc_server.log
- Each factorial calculation is logged with:
  - Input value.
  - Output result.
  - Errors.

→ Server Threading
- The server is started in a deamon thread.
- This lets it run in the background.
- The ~~with~~ while True: pass keeps the main program alive.

→ Security Note (Real-World Scenario)
- In real deployments:
  - Use authentication and encryption (HTTPS).
  - Avoid exposing the server to the internet without securing it.
- XML-RPC is ~~no~~ not encrypted by default use http:// and authentication headers if needed.

→ Advantages of RPC:-
- Simplifies remote interaction (like calling local function)
- Abstracts network code from developer.
- Language independent (client in one language, server in other).
- Good for modular and service-based ~~appl~~ applications.

→ Disadvantages of RPC :-
- Higher latency than local calls.
- Hard to debug (due to network failures or serialization errors).
- ~~Tight~~

☆ Code rpc-server.py

1) Imports
- SimpleXMLRPCServer: Creates a basic XML-RPC server to handle client calls.
- threading :- Used to run the server in a ~~sep~~ seperate thread (so it dosent block the main program).
- logging :- Used for maintaining logs of all server operations.

2) ~~Logging configuration:-~~

logging.basics....
- Sets up log file rpc-server.log.
- Logs each message with timestamps and message.
- Logs only INFO lever and above (includes errors too).

3) Function to compute factorial :-

- First checks if n is an integer & non-negative number.
- If not, it raises error.
- Else it computes factorial (from 1 to n).
- Stores in result.
- Logs the computation to the log file.
- returns the computed factorial to the client.

exception Exception as e:
. . .

- Logs an error that occurs during computation and returns the error string to the client for debugging.

start stop
(2, n+1))
↓ ↓
computing is not necessary — python range excludes stop value ∴ n+1.

4) function to start the XML-RPC server:
def start_server():
. . .

- starts a server on local-host 8000.
- allow_none = True :- allows 'None' values

to be sent or received. By default, the XML-RPC protocol does not allow None nor values (null) to be be passed between client & server. This makes server more robust under conditions like:-

- If error occurs and function decides to return none.
- If you extend this application later to return None under certain conditions (like database results).

- Register the factorial function so clients can ~~proxy~~ call ~~proxy.factorial()~~ rpc_client.py remotely using proxy.factorial().
when the server starts it ~~should~~ needs know which functions it should allow clients to use.
Without it server wouldn't know what to do when the client calls proxy.factorial().
- Server_forever() keeps the server alive and listening for client requests.

4) Run the server in a background thread

server_thread = ...

- Creates a new deamon thread to run the server function.
- A deamon thread stops when the main program ends.
- This allows the main program to stay responsive.

* A thread is a independent sequence of instruction that can be executed concurrently with other threads within the same process.

rpc_client.py :-

1) imports the xmlrpc.client module to connect to the server.

2) proxy = xmlrpc.client.ServerProxy(....)
   - creates a proxy object.
   - This objects lets you call remote functions (on the server) as if they were local.

3) Main function:-
   i) Asks the user to input a number.
   ii) Convats input to int.
   iii) Calls the server function: proxy.factorial(num)
      - Sends the request to server.
      - Server computs and returns the result.
   iv) Displays the result.

iv) Exception if the server is unreachable.
vi) Asks the user if they want to continue.

Real Life applications :-

1) Banking Systems
- Banks use centralized servers to calculate loans, credit scores etc
- Clients (ATMs, apps, website) send data to get results instantly.

2) Multiplayer games.
3) Any apps.
4) Online taxi portals.